

多媒體技術與應用 Spring 2021

Instructor : Yen-Lin Chen(陳彥霖), Ph.D.

Professor

Dept. Computer Science and Information Engineering

National Taipei University of Technology



Lecture 12

GAN生成對抗網路



GAN生成對抗網路-簡介

- GAN(Generative Adversarial Network，生成對抗網路)是2014年蒙特婁大學博士生 Ian Goodfellow 提出來的方方法。
- GAN的主要概念很簡單，以一個簡單的例子來做說明：
 - 在GAN裡面中有兩個角色，一個是專門偽造假名畫來去賣的**G先生**，一個是專門鑑定此符畫是否為真畫的**D先生**，D先生會從G先生那邊拿到假畫來辨斷真假，G先生則是利用D先生的鑑定來改良自己製造假畫的技術，不斷改良，最後讓假畫變成真假難辨。



GAN-簡介

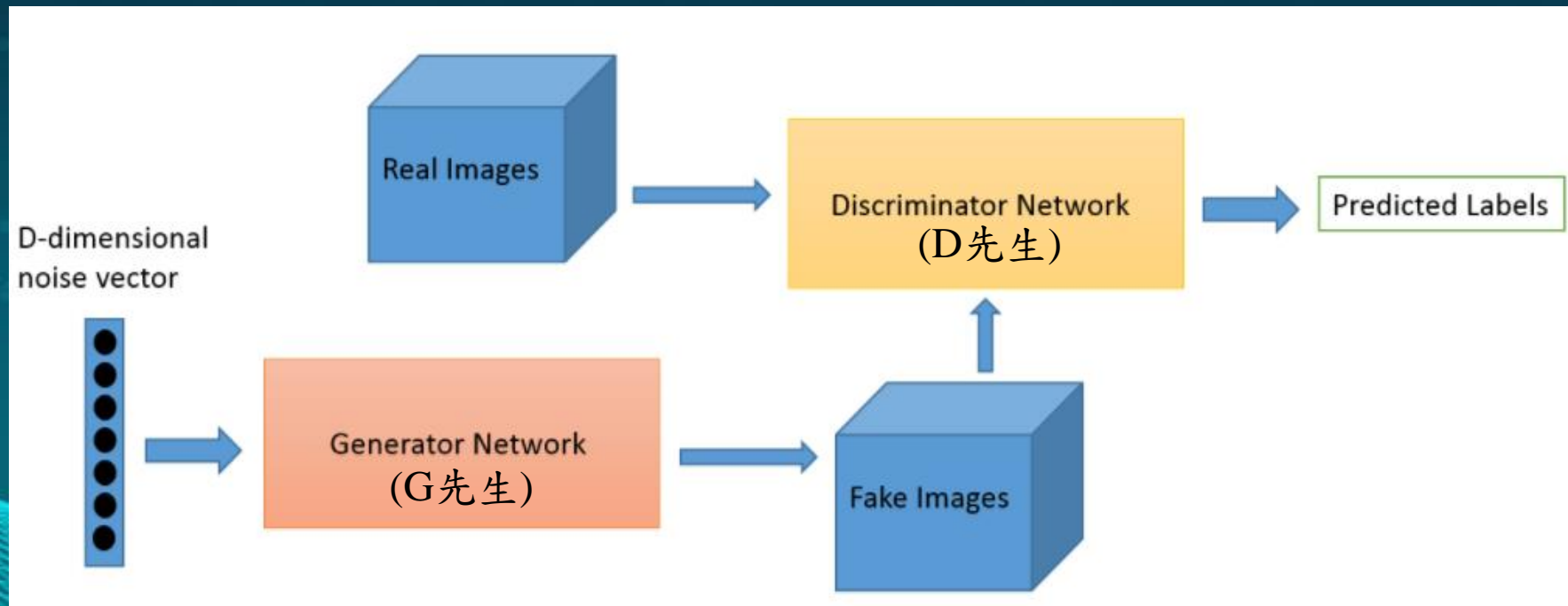
- 下圖中，看似只是個溫馨的寢室照片，實際上這些都是GAN所製造出來的假圖。





GAN-簡介

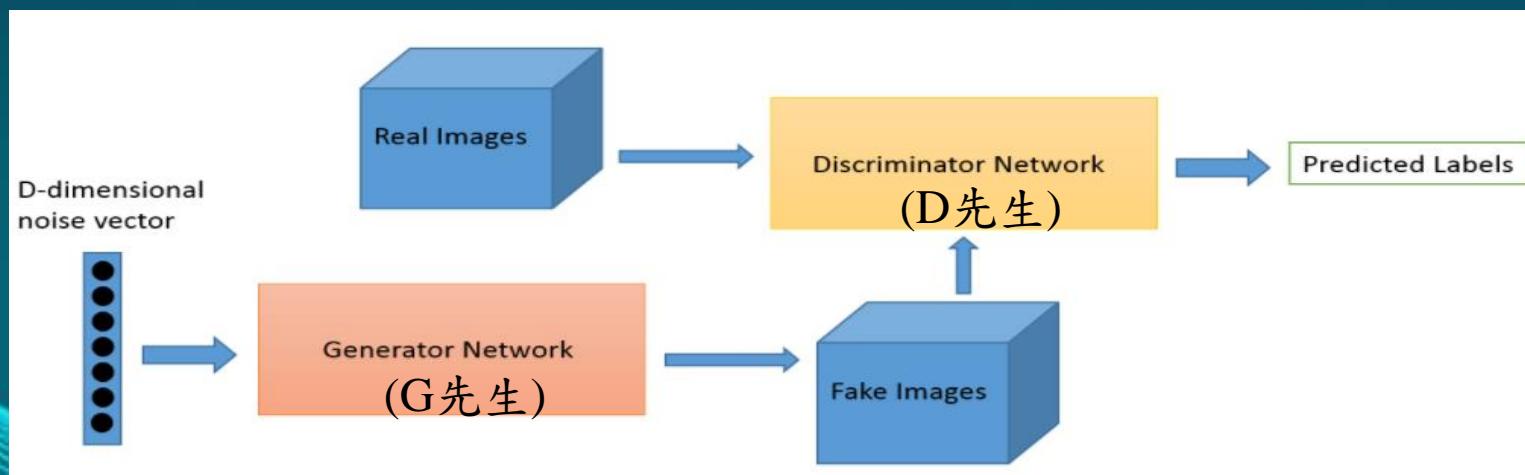
- 在GAN架構下，偽造者就稱為『生成模型』（Generative model），鑑定者稱為『判別模型』（Discriminative model）（這邊的G與D就是對應先前例子所提到的G先生與D先生），簡單架構如下圖：





GAN-簡介

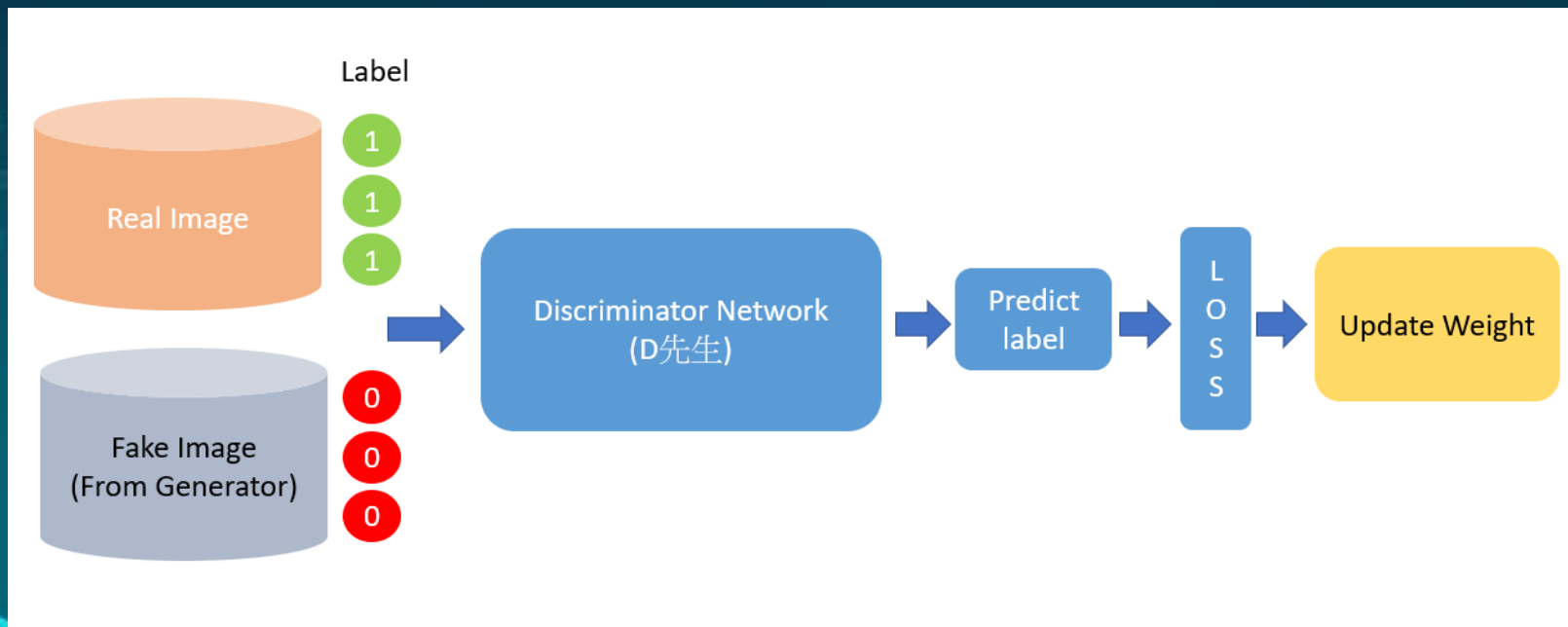
- 圖中，生成模型利用已知的真畫加上雜訊(Noise)來製造假畫，交給判別模型辨識，它是二元分類模型，只會判斷真或偽，再將結果回饋給生成模型，經過不斷的訓練，就生成愈來愈像的樣本了。
- 圖中是一個最基本的GAN的流程圖，可以看出GAN中有二個Neural Network需要去訓練，接下來將會介紹Discriminator跟Generator這二個神經網路應該要怎麼去訓練。





Discriminator Network(判別網路)

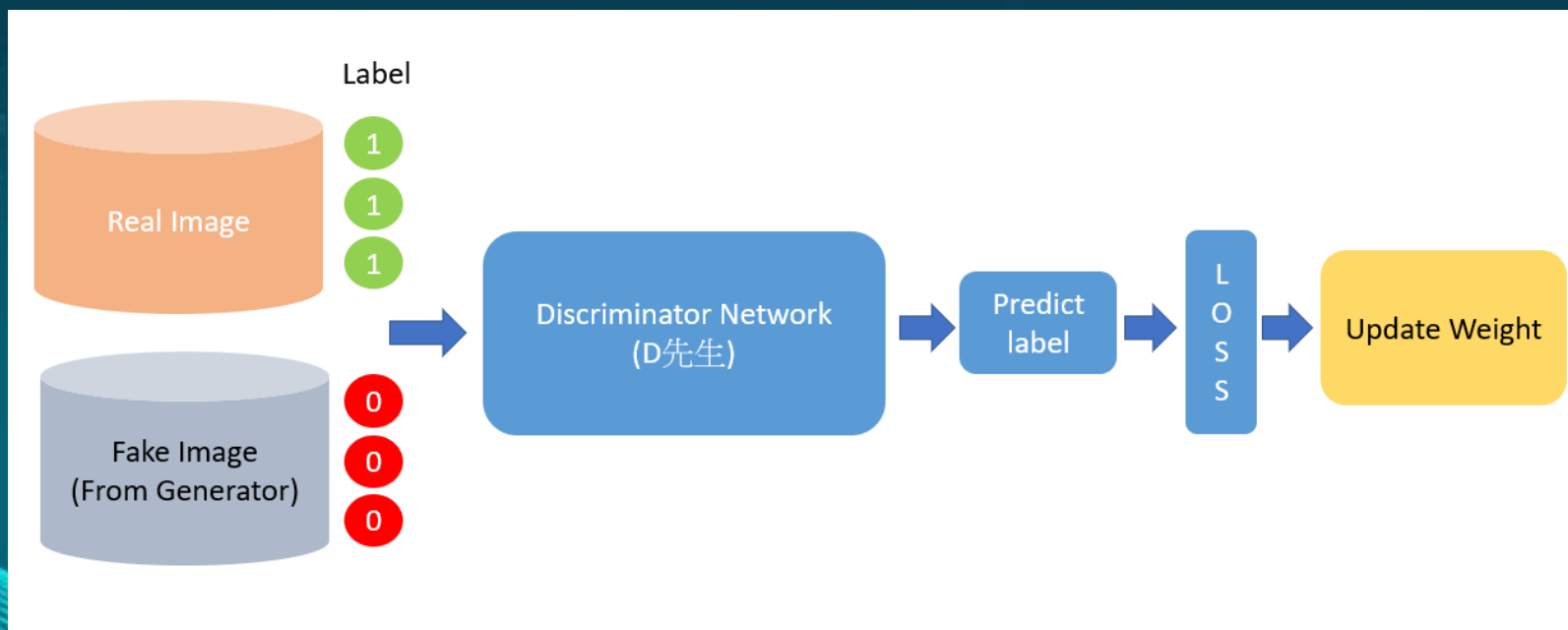
- 判別網路簡單一點說明的就是，訓練出一個Neural Network可以分辨偽造出來的圖跟真實的圖，方法可以用以下的圖來理解。





Discriminator Network(判別網路)

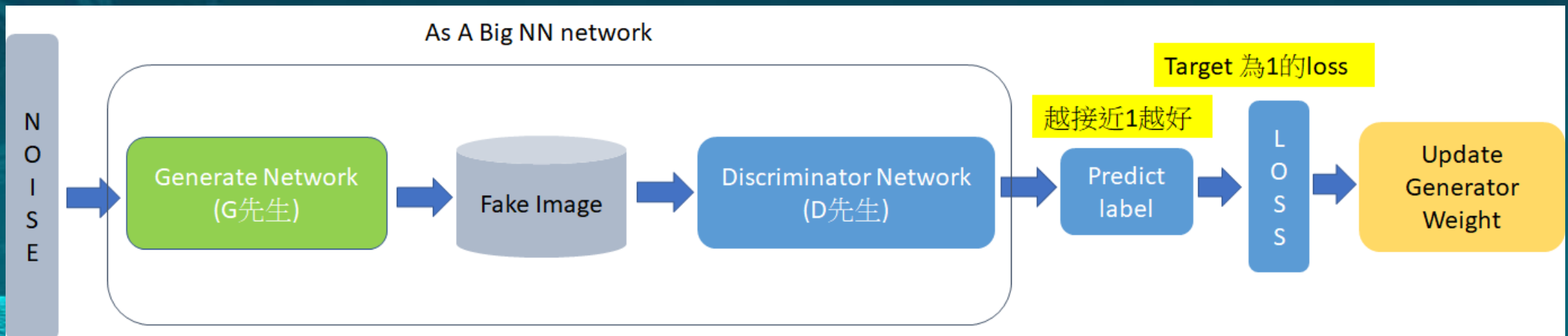
- 很直觀的，可以直接把Generator Network(生成網路)產生出來的圖標記為0(fake image，假貨)，然後把真實的圖標記為1，這樣的training data 丟進我們的Discriminator Network做訓練，這就是每一次Discriminator訓練的步驟了。





Generator Network (生成網路)

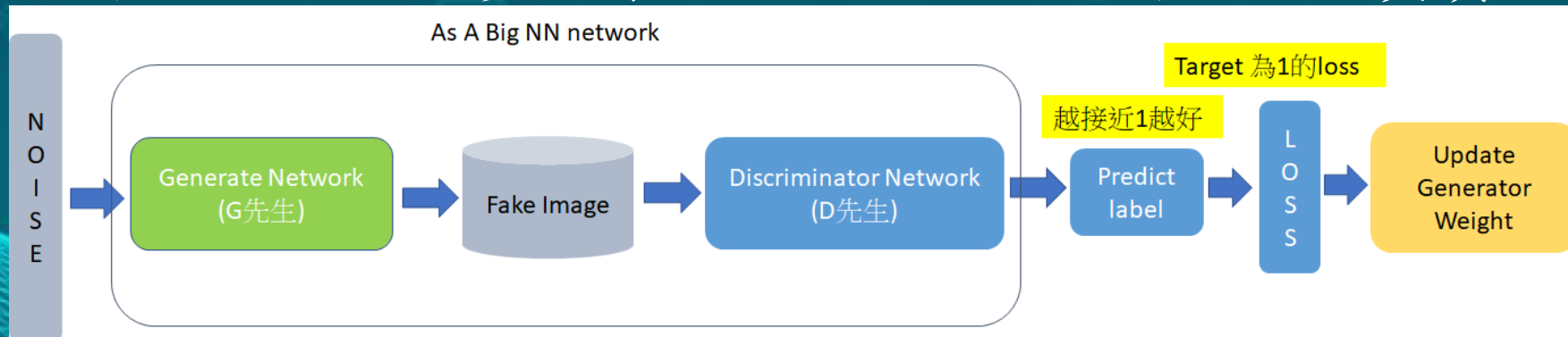
- 生成網路的概念也很簡單，就是要訓練出一個Neural Network可以產生出讓Discriminator分辨出來的結果愈接近真實(1)的結果愈好(換句話說，就是讓Discriminator無法正確的分辨Generator所產生的圖片是真是假)，可以用以下的圖來理解生成網路的訓練方式。





Generator Network (生成網路)

- 我們可以把Generator + Discriminator看成是一個大的Neural Network，假設生成網路是前5層的Neural，判別網路是後5層的Neural，然後讓這個10層的Neural Network 預估出來的值愈接近1(真實)愈好，但是這裡我們只更新 Generator 的權重值，而Discriminator的權重值則要保持不變，這樣才可以用更新後Generator所產生出來的假圖讓Discriminator進行測試，輸出的值愈接近真實的結果代表Generator的訓練愈成功，簡單來說，其實就是更新Generator的參數讓Discriminator預測出接近真實的結果。





GAN演算法-說明

- GAN基本分為Generator 與 Discriminator兩部分。
- G、D之間的關係會是一種minmax game的關係(零和遊戲，在一項遊戲中，遊戲者有輸有贏，一方贏就代表另一方輸，遊戲的總成績永遠為零)(minmax演算法，一方要在可選的選項中選擇將其優勢最大化的選擇，另一方則選擇令對手優勢最小化的方法。)



GAN演算法-說明

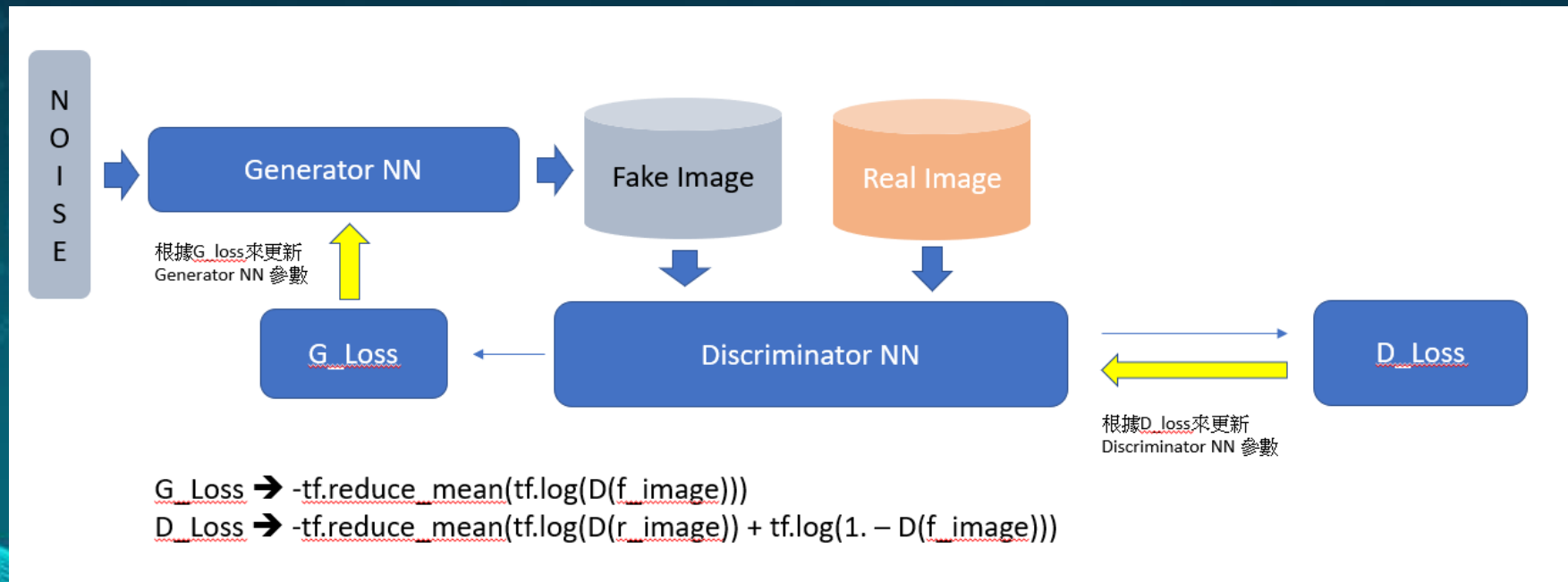
- GAN的目標函數(即損失函數loss function)就是下方圖中的公式，是G與D的目標函數加總，D要從G所生成的影像判斷真偽，G則是想盡辦法要騙過D，所以，G、D目標函數是兩個反向的 $\log(D(x))$ 與 $\log(1 - D(G(z)))$ ， x 是真實的影像， z 是雜訊(Noise)，假設 z 是按照『均勻分布』(Uniform distribution) 或是『常態分佈』(Normal distribution or Gaussian distribution)，最佳化目標是使G的機率分布趨近於D的機率分布，也就是使生成的影像(G產生的影像)盡可能接近真實的影像(訓練D時的正確影像)，求解的方法則是用『最大概似估計』(Maximum likelihood Estimation，MLE)。

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$$



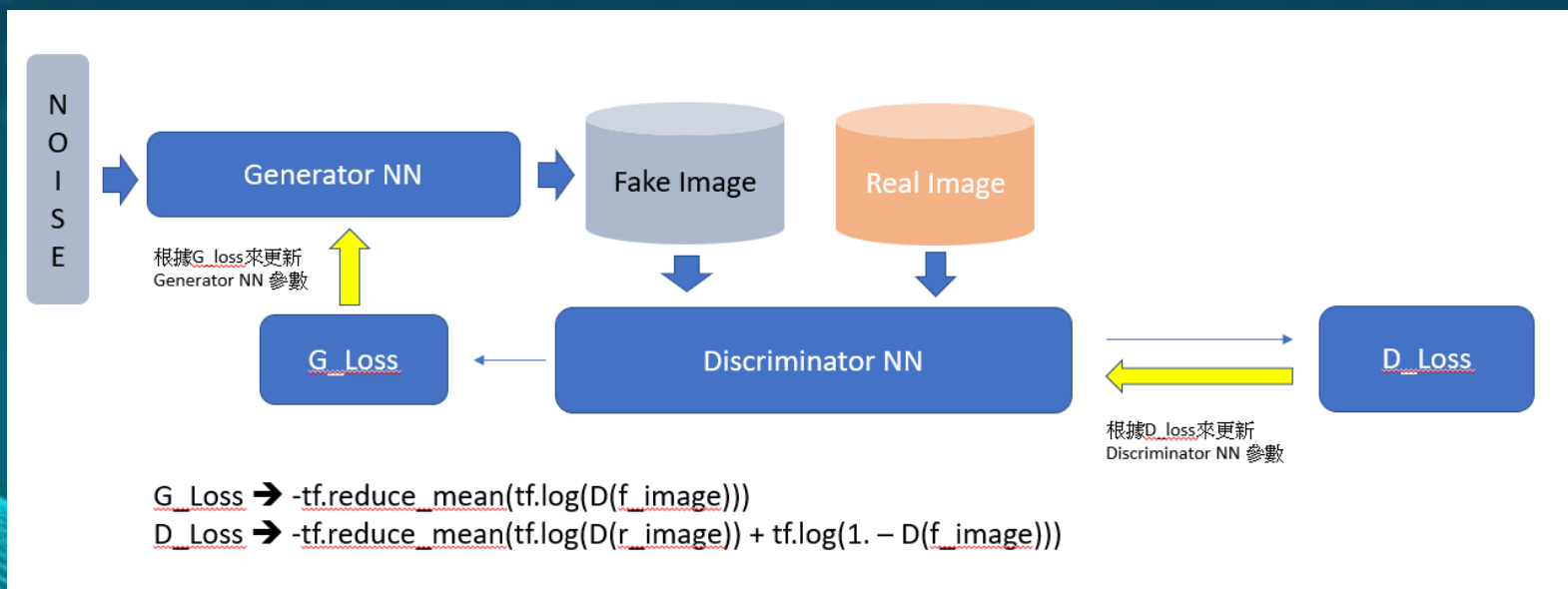
GAN演算法-說明

- 依照演算法上的Loss function，整理了整個GAN的流程，可以更容易了解GAN的基本運作。



GAN演算法-說明

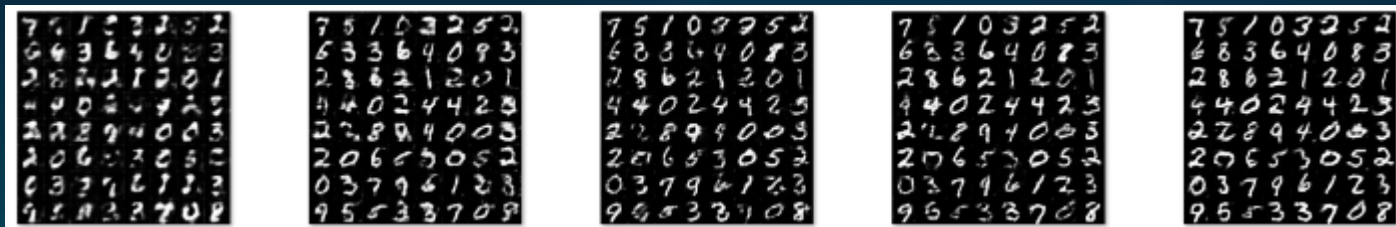
- G_Loss就是我們把Generator產生的偽造圖輸入進Discriminator中的輸出與1的loss
- D_Loss就是我們把Generator產生的偽造圖輸入進Discriminator中的輸出與0的loss + 真實的圖輸入進Discriminator中的輸出與1的loss。





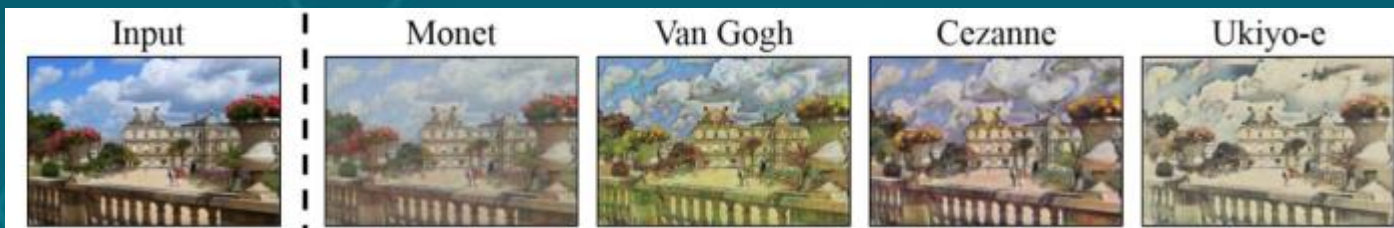
GAN-各式應用

- 不用再花大量人力標註資料，直接利用GAN模型生成即可，『監督式學習』(Supervised)就變成『非監督式學習』(Unsupervised)。



圖片來源：[Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks](#)

- 風格轉換(Style Transfer)：透過生成，就可以輕易把梵谷畫風轉移到另一張照片上了。此外，也可以反過來，把梵谷畫作轉換成照片。



圖片來源：[Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks](#)



GAN-各式應用

- 高解析度影像生成：透過不斷的生成與判別，模型最後可以訓練出比原圖更高解析度的影像。



圖片來源：[Tutorial on Deep Generative Models](#)



GAN-各式應用

- 壓縮影像：在網路傳輸大量影像時，為節省頻寬，常需要壓縮影像，也可以利用GAN生成。

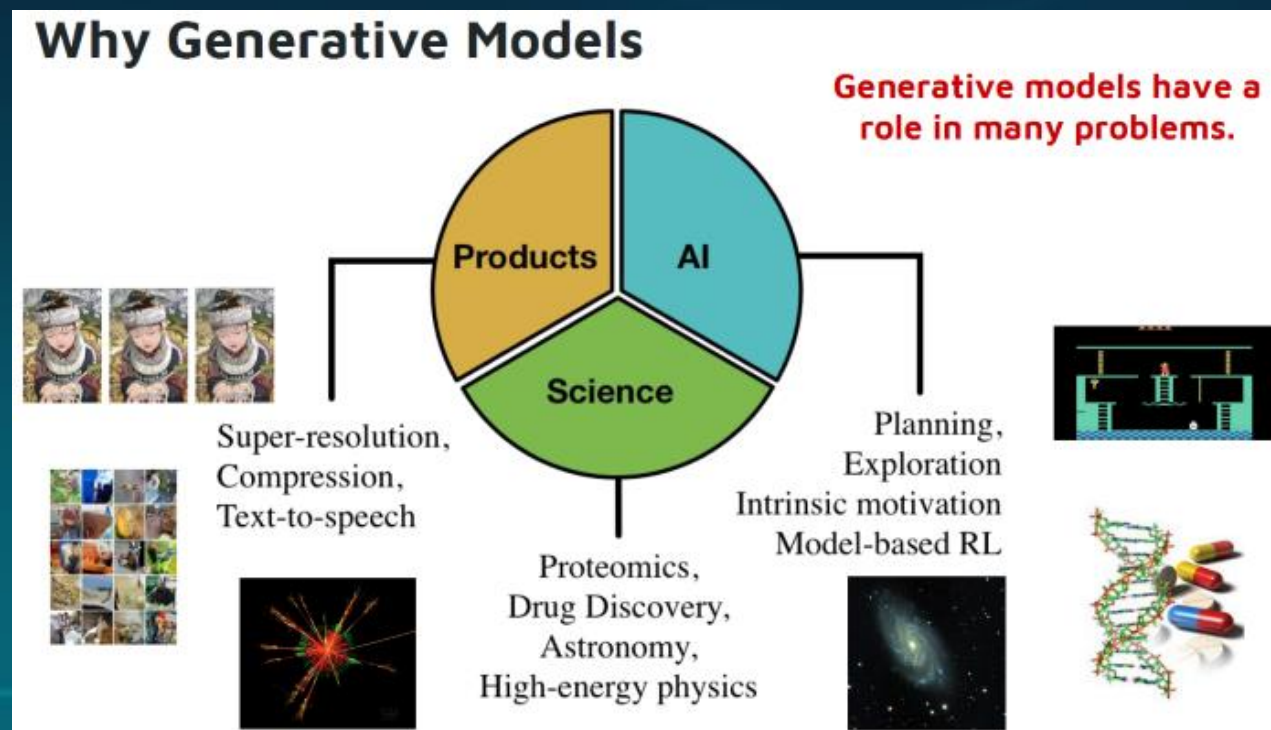


圖片來源：[Tutorial on Deep Generative Models](#)



GAN-各式應用

- 還有醫學、天文等等領域的應用，GAN可以做到的事情非常多。

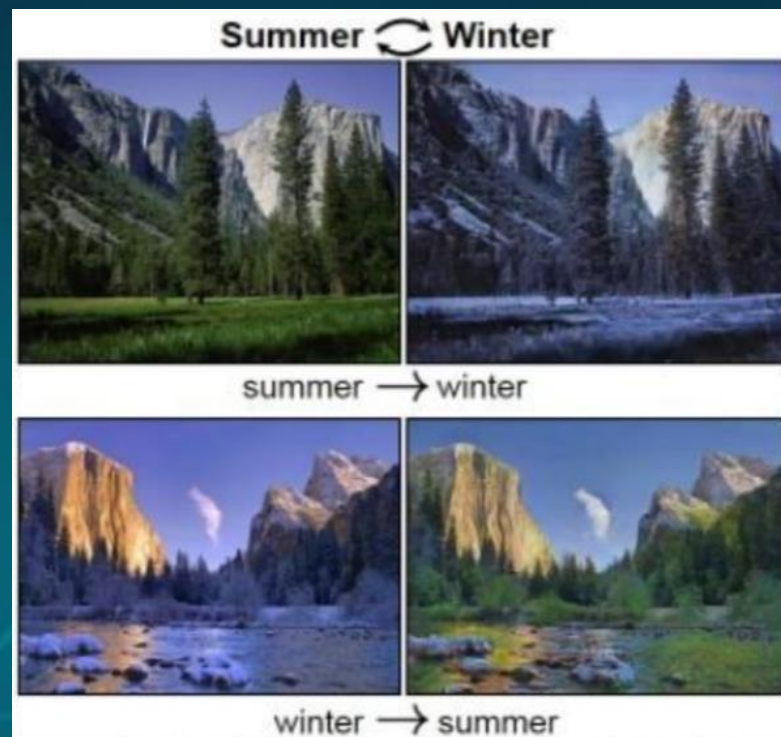
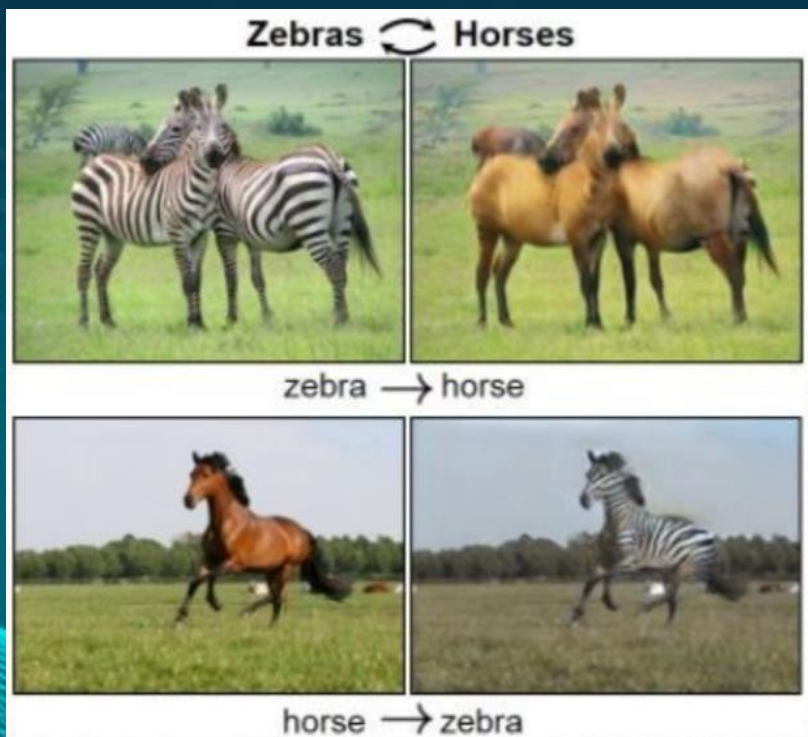


圖片來源：[Tutorial on Deep Generative Models](#)



GAN-社會實例應用說明

- 前面的例子有提到過，利用GAN，可以做到在相似形體的物件上，套用另一種物件的特徵，像是馬與斑馬，或是季節變化等等。
- 那麼，是否有可能利用GAN的技術，來做到換臉這件事情呢？





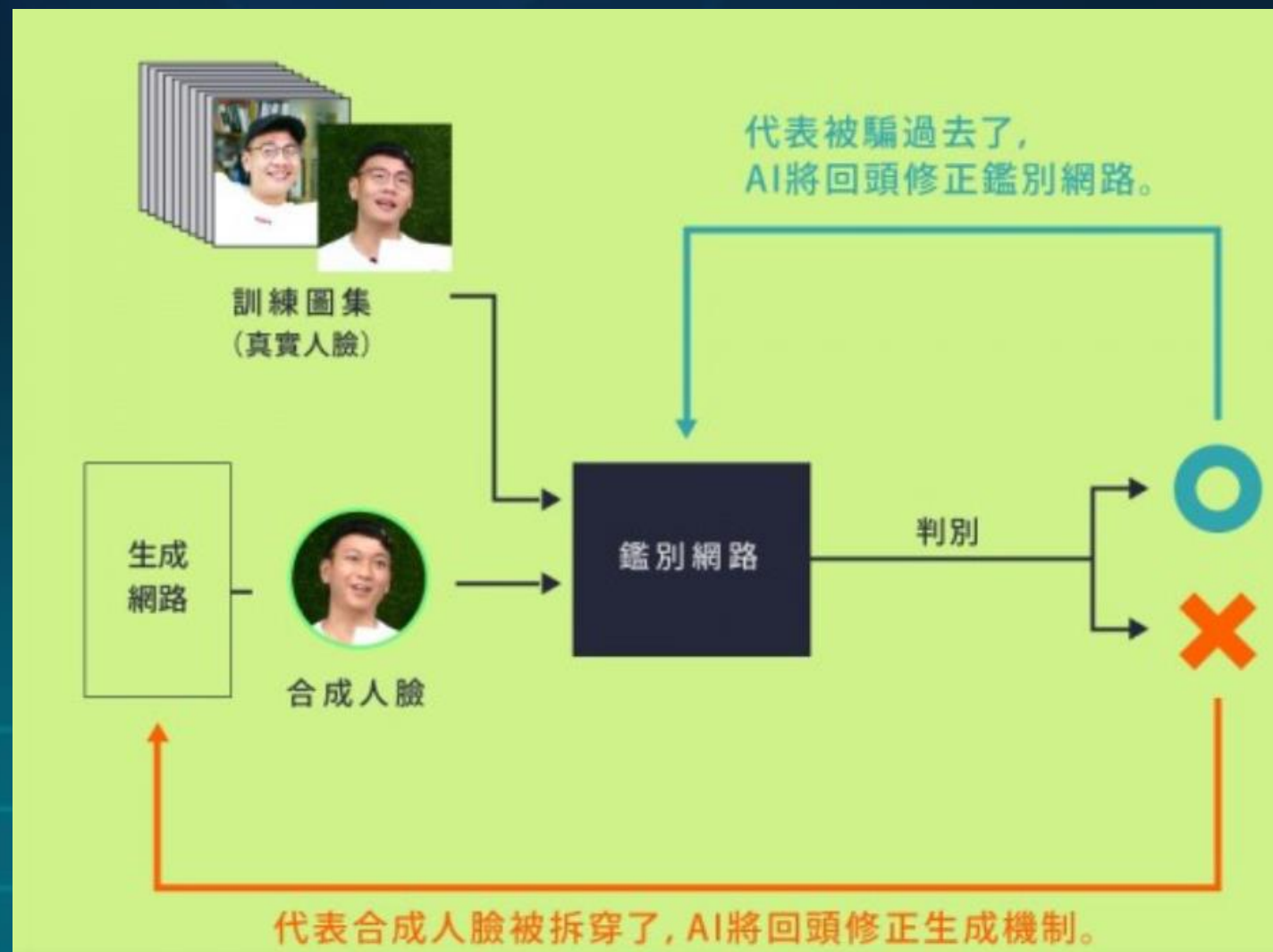
GAN-DeepFake

- 深偽技術（英語：Deepfake）又稱深度偽造，是英文「deep learning」（深度學習）和「fake」（偽造）的混合詞，專指基於人工智慧的人體影像合成技術的應用。此技術可將已有的影像或影片疊加至目標影像或影片上。
- Deepfake的生成一般會有兩種方法。一個叫做自編碼器（**AutoEncoder**，為在Social LSTM課程中所提到的，Encoder與Decoder技術的一項應用），另一個便是本周的主題生成對抗網路（**Generative Adversarial Network, GAN**），兩者都是深度學習的應用方式。



GAN-DeepFake

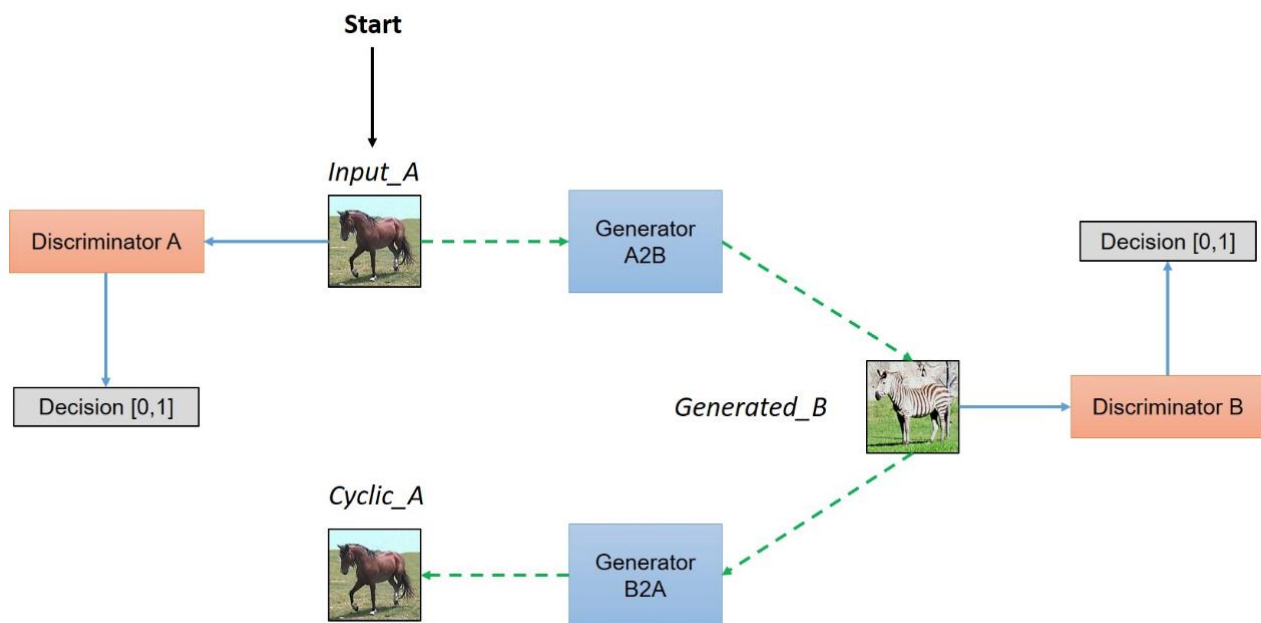
- 做法基本與傳統的GAN相同，以Generator生成假的影像，使一個人的臉疊加在另一個人臉上，並交由Discriminator進行判別是否為真實的，圖中藍色線條的方向會強化Discriminator的辨識能力，紅色線條的方向則是使Generator仿造得更真實。





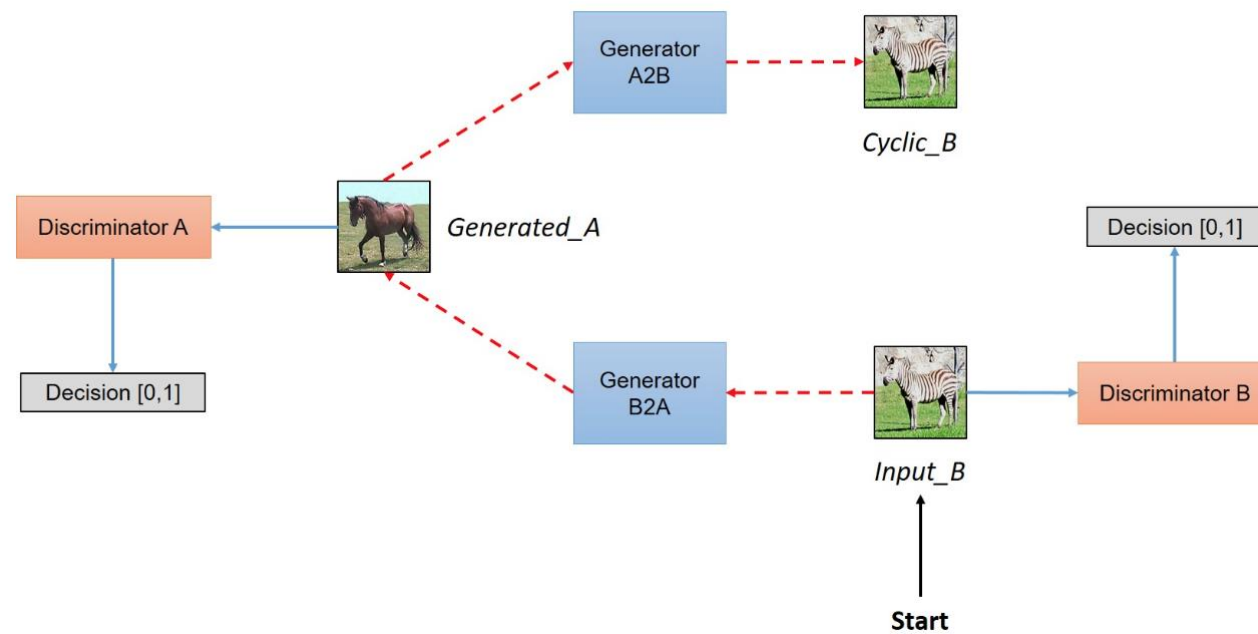
GAN- Cycle GAN

- 因為GAN是非監督式學習，無法有效的確保模型效果，因此產生了一種進階模型 – Cycle GAN，將兩個不同domain的資料進行轉換。
- 以馬與斑馬的例子來說明，Cycle GAN 的想法是將 A domain (例如馬) 的影像轉成 B domain (例如斑馬)，因為是非監督式學習所以不好保證效果，但可以再從 B domain (例如斑馬) 的圖轉回 A domain (例如馬)，轉回的還原圖要跟原本給的 input 越像越好



把 A(馬)轉成 B(斑馬)再轉回 A(馬)

把 B(斑馬) 轉成 A(馬) 再轉回 B(斑馬)





GAN-社會實例應用說明-DeepFake

- DeepFake的實際應用效果





GAN-社會實例應用說明-DeepFake

- DeepFake背後帶來的問題：
 - 政治：利用DeepFake模仿政治家發表不適當言論。
 - 色情：將知名女星的頭附加在不適當的身體上。
- 使用者不需要知道其背後的原理，只要小小的成本，就可以購買到完整的應用程式，也可以自行尋找open source code進行使用。



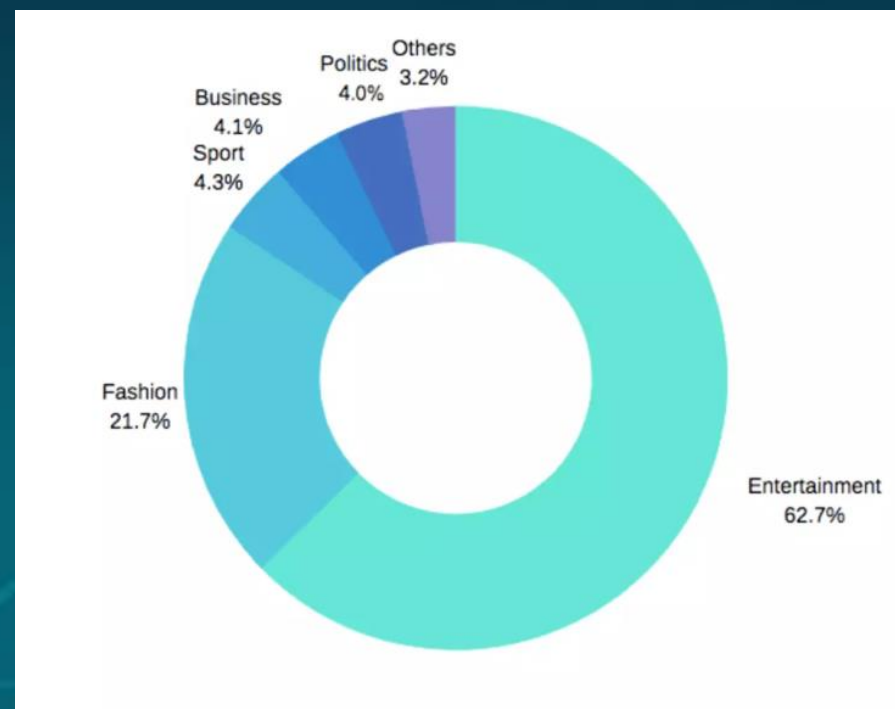
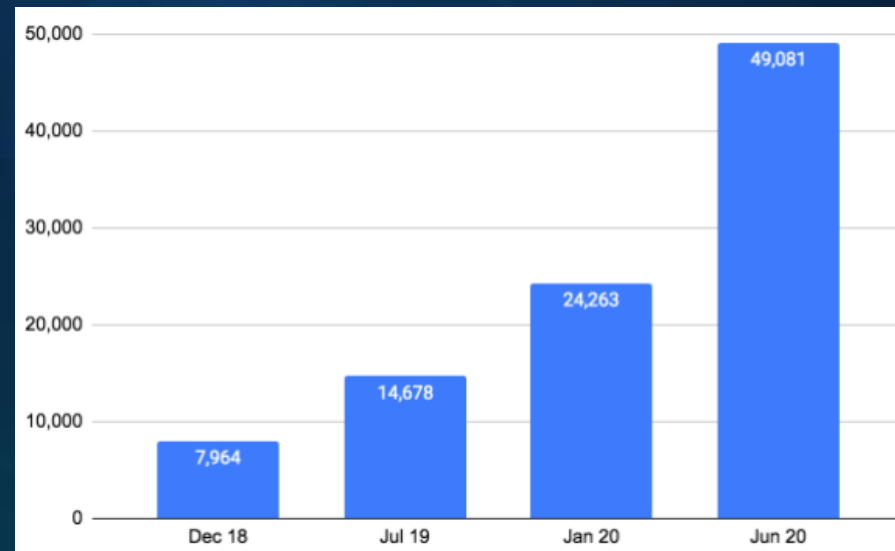
<https://www.youtube.com/watch?v=cQ54GDm1eL0>





GAN-DeepFake

- DeepFake背後帶來的問題：
 - 在美國2018年12月以來，Deepfake 網路造假品數量約每6個月翻一倍。
 - 2019年，14,678個公開的Deepfakes作品，假色情影片就占96%。
 - 2020年，Deepfake製作色情作品的目標對象影視娛樂是涵蓋最廣的行業，占62.7%，加上時尚類別（21.7%）和運動類別（4.4%），總計占所有目標88.9%。

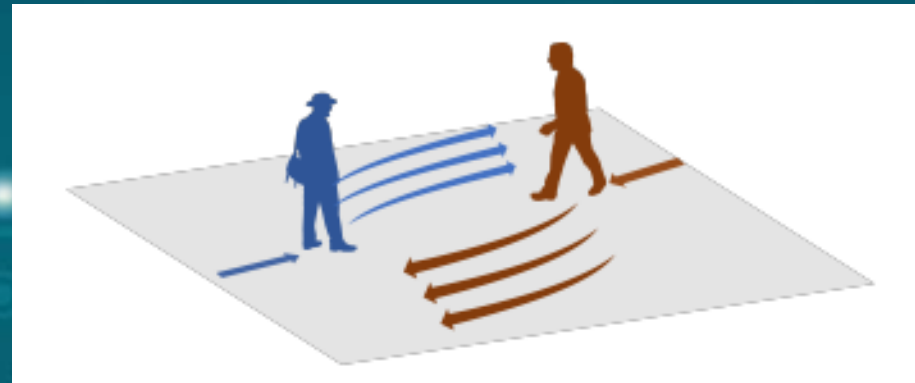




Social GAN

與Social LSTM同為軌跡預測模型，但Social GAN提出針對以下幾點進行改良：

- Interpersonal：行人在移動時要考慮自己的移動方向外，同時也要考慮到周圍其他人的移動方向
- Socially Acceptable：預測軌跡除了要符合實際路線外，也需要符合社交禮儀
- Multimodal：在符合上述兩點的情況下，可以產生的路徑可能是非單一的結果





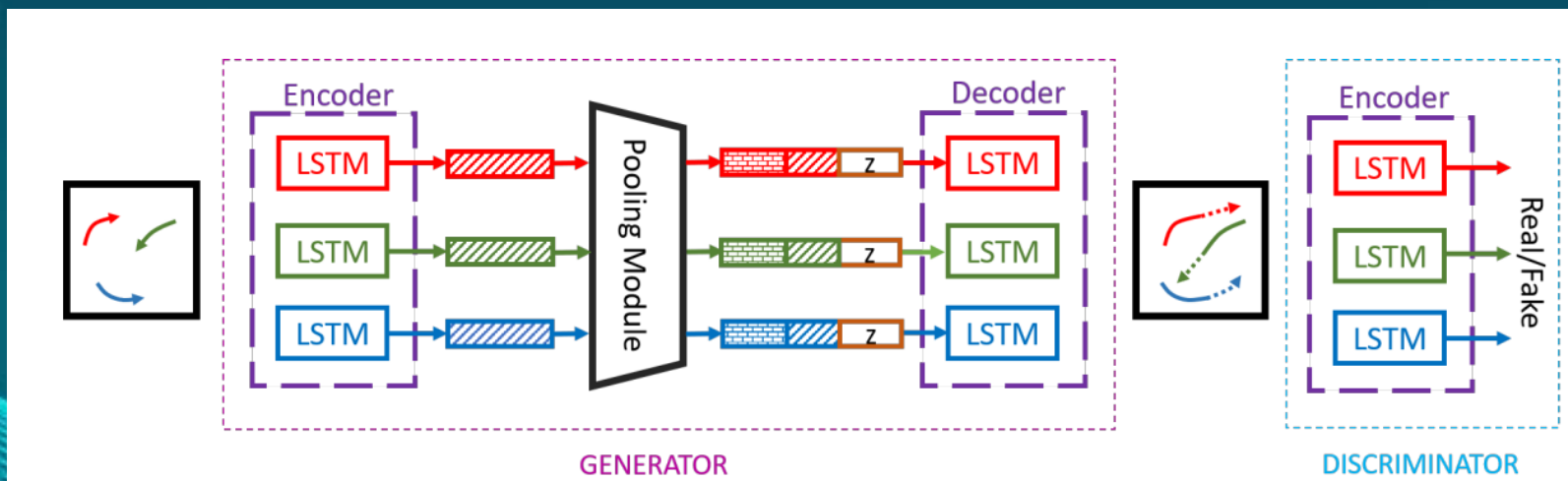
Social GAN-軌跡預測的問題定義

- 將場景中的所有交通物件軌跡輸入
 - $X = X_1, X_2, \dots, X_n$, X_i 代表第 i 個物件的軌跡
 - 第 i 個交通物件的軌跡輸入為 $X_i = (x_i^t, y_i^t)$, 時間 $t = 1, \dots, t_{obs}$
- 輸出所有交通物件的未來預測軌跡
 - $\hat{Y} = \hat{Y}_1, \hat{Y}_2, \dots, \hat{Y}_n$
 - 第 i 個交通物件的預測軌跡輸出為 $\hat{Y}_i = (\hat{x}_i^t, \hat{y}_i^t)$, 時間 $t = t_{obs} + 1 \dots, t_{pred}$
- 真實軌跡的定義為：
 $Y_i = (x_i^t, y_i^t)$, 時間 $t = t_{obs} + 1 \dots, t_{pred}$



Social GAN-架構

- Socially-Aware GAN
 - Generator：輸入過去的軌跡資料，輸出預測的軌跡資料
 - Discriminator：輸入預測的軌跡資料，並判斷其是Real或Fake
- Encoder-Decoder
 - Encoder：透過LSTM對歷史資料取出每條軌跡的Hidden State
 - Decoder：將Pooling Module輸出的社交關係與Encoder輸出的Hidden State，結合雜訊 z 產生預測軌跡
- Pooling Module
 - 輸入同一畫面中所有軌跡的座標與Encoder的hidden state，並輸出物件與其他物件之間的交互關係





Social GAN-Encoder

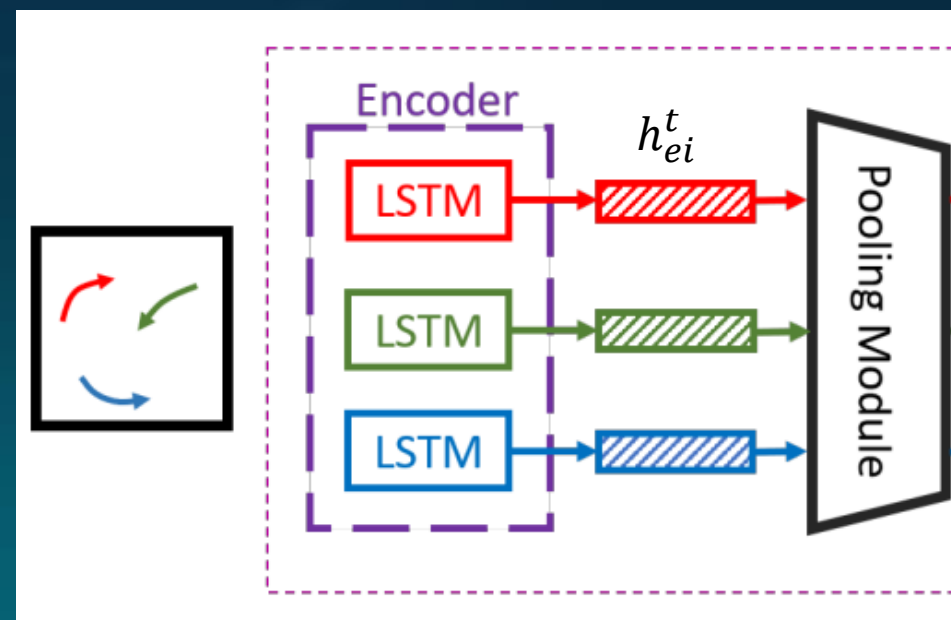
將input 資料轉為固定長度的Vector

- $e_i^t = \varphi(x_i^t, y_i^t; W_{ee})$
 - e_i^t 為固定長度的向量
 - $\varphi()$ 為使用ReLU的embedding函數
 - W_{ee} 為embedding的權重

將Embedding後的向量輸入LSTM

- $h_{ei}^t = LSTM(h_{ei}^{t-1}, e_i^t; W_{encoder})$
 - $W_{encoder}$ 為LSTM的權重

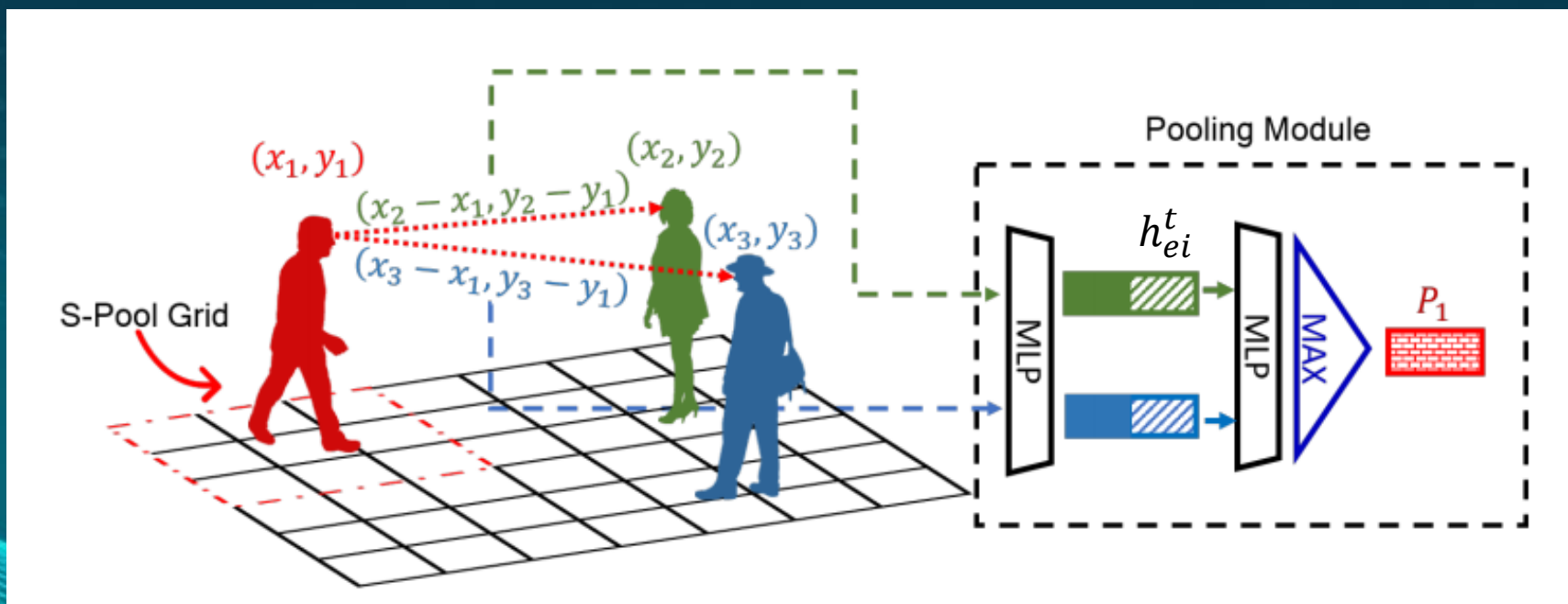
最後得到包含過去軌跡的資訊 h_{ei}^t





Pooling Model

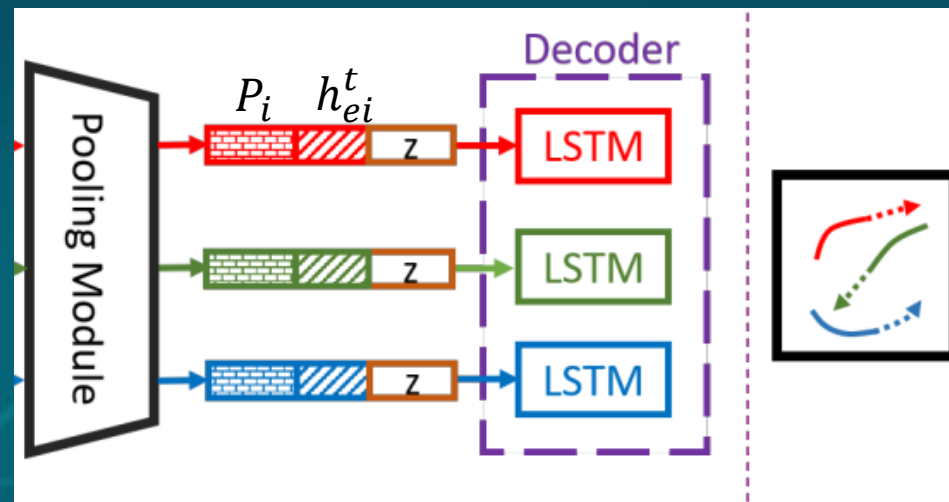
- 將目標與周遭物件的距離傳入MLP並輸出向量
- 將第一次MLP的結果與Encoder的輸出結合，再透過MLP與Max-Pooling得出物件與物件間的社交互動關係 P_i





Social GAN-Decoder

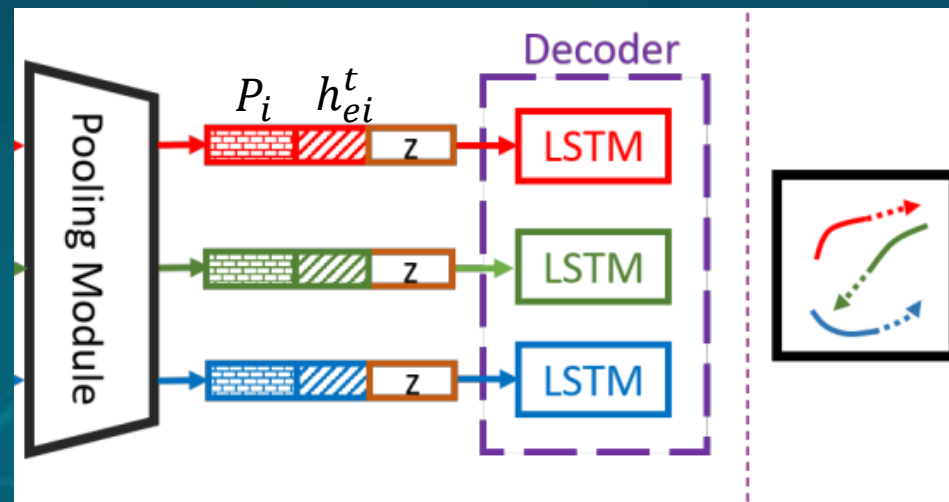
- 透過初始化Decoder的hidden state來調整軌跡的生成
- $c_i^t = \gamma(P_i, h_{ei}^t; W_c)$
 - $\gamma()$ 為帶有ReLU的MLP
 - W_c 為embedding的權重
- 初始化後的hidden state用以輸入LSTM中
- 將 c_i^t 與雜訊 z 串聯成 $h_{di}^t = [c_i^t, z]$





Social GAN-Decoder (cont.)

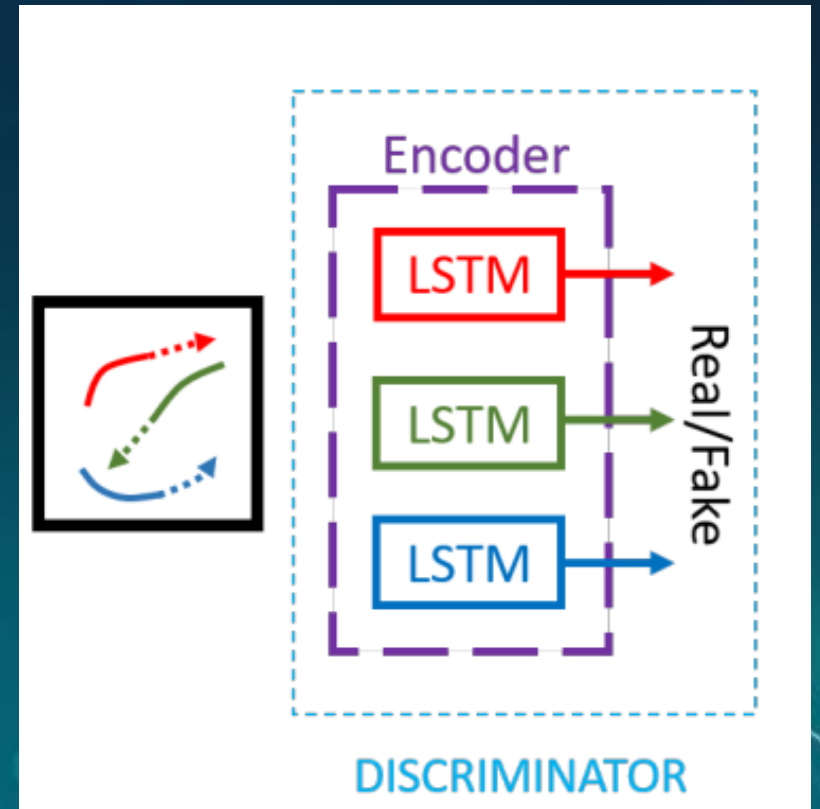
- $e_i^t = \varphi(x_i^{t-1}, y_i^{t-1}; W_{ed})$
 - φ 為帶有ReLU之embedding 函數
 - W_{ed} 為embedding 權重
- $P_i = PM(h_{d1}^{t-1}, \dots, h_{dn}^t)$ ，為Pooling Module的輸出
- $h_{di}^t = LSTM(\gamma(P_i, h_{di}^{t-1}), e_i^t; W_{decoder})$
 - $W_{decoder}$ 為 LSTM 權重
 - γ 是 MLP 函數
- 輸出預測座標 $(\hat{x}_i^t, \hat{y}_i^t) = \gamma(h_{di}^t)$
 - γ 是 MLP 函數





Social GAN-Discriminator

- 將預測的軌跡輸入Discriminator
- 並透過LSTM將所有路徑的hidden State輸出
- 最後以MLP作用於Encoder的Hidden State，得到真假的判別分數





Social GAN-Loss

- 在訓練Generator前我們需要先訓練Discriminator，使用的是目標函數：
 - $\min_G \max_D V(G, D) = E_{x \sim P_{data}(x)} [\log D(x)] + E_{z \sim p(z)} [\log(1 - D(G(z)))]$
 - x 為真實數據的資料，因此要給他高分
 - z 為Generator配合亂數所產生的資料，因此要給他低分
- 為了讓Generator可以多去搜尋其他合理的路徑，定義預測多樣性軌跡的損失函數：
 - $L_{variety} = \min_k \left\| y_i - \hat{y}_i^{(k)} \right\|_2$
 - 預測出 k 條不同的軌跡
 - 再以最接近真實軌跡的預測軌跡來進行backpropagation



專案下載

- (本專案推薦使用Google Colab雲端計算平台，也可使用自己的電腦進行訓練)
- 請到以下網頁下載：
 - <https://drive.google.com/drive/folders/1pJnv8jo-BDqFDGpsDtlrcxQkdGMnFA1z?usp=sharing>
- 若使用Google Colab，先將檔案上傳至自己雲端後，再掛載雲端到colab：



```
#掛載自己的google雲端硬碟|  
from google.colab import drive  
drive.mount('/content/drive')
```



專案下載

- 將掛載資料夾中的檔案copy到Colab工作空間：

```
!cp -r /content/drive/MyDrive/多媒體助教/GAN/SocialGAN2 /content
```


- 之後進入資料夾：
 - cd SocialGAN

```
%cd /content/SocialGAN/
```



訓練模型


- 在終端機中輸入指令即可訓練模型：
`python3 train.py`



```
!python train.py
```

- 若在訓練過程遇到以下情況，可以將torch版本降至1.4.0版

```
torch.Size([8, 87, 2])
Traceback (most recent call last):
  File "train.py", line 922, in <module>
    main(args)
  File "train.py", line 564, in main
    optimizer_d)
  File "train.py", line 620, in discriminator_step
    generator_out = generator(obs_traj, obs_traj_rel, typeID_seq, seq_sta
  File "/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py
    result = self.forward(*input, **kwargs)
  File "/content/SocialGAN2/sgan/models.py", line 513, in forward
    mlp_decoder_context_input = final_encoder_h.view(
AttributeError: 'tuple' object has no attribute 'view'
```



```
!pip install torch==1.4.0
```




train.py 訓練參數介紹

- --embedding_dim: 嵌入參數的維度
- --encoder_h_dim_g: Encoder 中 Hidden State 的長度
- --decoder_h_dim_g: Decoder 中 Hidden State 的長度
- --g_learning_rate: 每次訓練時調整 Generator 的幅度
- --g_steps: 每次訓練時 Generator 的個別訓練次數
- --encoder_h_dim_d: Encoder 中 Hidden State 的長度
(判別器)
- --d_learning_rate: 每次訓練時調整 Discriminator 的幅度
- --d_steps: 每次訓練時 Discriminator 的個別訓練次數

皆可依據小組需求
自行更改訓練參數
以達到不同的訓練
結果。



train.py 訓練參數介紹

- `--print_check`: 每幾次進行測試
- `--checkpoint_every`: 每幾次存一次模型權重檔
- `--checkpoint_name`: 模型權重檔名，
會自動加上“_with_model.pt”
- `--use_gpu`: 使用GPU及CUDA函式庫進行訓練
- `--logname`: 將訓練進度輸出成文字檔的檔名
- `--use_gru`: 使用GRU或LSTM架構，1為GRU，0為LSTM

皆可依據小組需求自行更改訓練參數以達到不同的訓練結果。

除上述所介紹之參數外，其他許多參數為尚待實作的功能，自行設定可能導致程式無法運行

※加上訓練參數的範例：



```
python train.py --num_epochs 200 --use_gpu 1
```



訓練過程

- 程式會在每次訓練過程中顯示損失函數的值及每個封包(batch)執行的平均時間

```
python train.py

[WARNING: train.py: 173]: Initializing train dataset: 04171525
[INFO: train.py: 175]: Initializing val dataset
[INFO: train.py: 396]: There are 4453 iterations in total, 8.90625 iterations per epoch
[INFO: train.py: 553]: Starting epoch 1, 2021-06-09 17:49:28.889771
[INFO: train.py: 584]: t = 1 / 4453
[INFO: train.py: 586]: [D] D_data_loss: 3.015
[INFO: train.py: 586]: [D] D_total_loss: 3.015
[INFO: train.py: 589]: [G] G_discriminator_loss: 0.469
[INFO: train.py: 589]: [G] G_l2_loss_rel: 4.438
[INFO: train.py: 589]: [G] G_total_loss: 4.908
[INFO: train.py: 553]: Starting epoch 2, 2021-06-09 17:50:23.450836
[INFO: train.py: 553]: Starting epoch 3, 2021-06-09 17:51:18.318425
[INFO: train.py: 553]: Starting epoch 4, 2021-06-09 17:52:14.132634
[INFO: train.py: 584]: t = 21 / 4453
[INFO: train.py: 586]: [D] D_data_loss: 1.386
[INFO: train.py: 586]: [D] D_total_loss: 1.386
[INFO: train.py: 589]: [G] G_discriminator_loss: 0.693
[INFO: train.py: 589]: [G] G_l2_loss_rel: 0.224
[INFO: train.py: 589]: [G] G_total_loss: 0.917
[INFO: train.py: 503]: Checking stats on val ...
[INFO: train.py: 507]: Checking stats on train ...
[INFO: train.py: 514]: [val] ade: 0.654
[INFO: train.py: 514]: [val] ade_l: 0.944
[INFO: train.py: 514]: [val] ade_nl: 2.128
[INFO: train.py: 514]: [val] d_loss: 1.386
[INFO: train.py: 514]: [val] fde: 1.121
[INFO: train.py: 514]: [val] fde_l: 1.618
[INFO: train.py: 514]: [val] fde_nl: 3.648
[INFO: train.py: 514]: [val] g_l2_loss_abs: 0.213
[INFO: train.py: 514]: [val] g_l2_loss_rel: 0.213
[INFO: train.py: 517]: [train] ade: 0.514
[INFO: train.py: 517]: [train] ade_l: 0.992
[INFO: train.py: 517]: [train] ade_nl: 1.067
[INFO: train.py: 517]: [train] d_loss: 1.386
[INFO: train.py: 517]: [train] fde: 0.874
[INFO: train.py: 517]: [train] fde_l: 1.687
[INFO: train.py: 517]: [train] fde_nl: 1.814
[INFO: train.py: 517]: [train] g_l2_loss_abs: 0.139
[INFO: train.py: 517]: [train] g_l2_loss_rel: 0.139
[INFO: train.py: 524]: New low for avg_disp_error
[INFO: train.py: 530]: New low for avg_disp_error_nl
[INFO: train.py: 544]: Saving checkpoint to /content/SocialGAN2/Yangde_every_with_model.pt
[INFO: train.py: 546]: Saving process is done.
[INFO: train.py: 553]: Starting epoch 5, 2021-06-09 17:53:17.492752
```




模型驗證

- 在終端機中輸入指令即可進行驗證、獲得loss、ADE、FDE等值:
`python3 evaluate_model.py --model_path <模型檔案路徑>`

```
+ 程式碼 + 文字
!python evaluate_model.py --model_path "./Yangde_every_with_model.pt"

Model path: ./Yangde_every_with_model.pt
Dataset: 04171525, Pred Len: 8, ADE: 0.15, FDE: 0.28, Time: 25.77160 (s), Counts: 6900, Avg time: 0.00374
```



模型誤差計算

在軌跡預測任務中，常用來驗證模型誤差的誤差測量有以下兩種：

- 平均位移誤差(Average Displacement Error, ADE)：
以每個預測點與真實點的平均歐式距離做為整條軌跡的平均誤差
- 終點位移誤差(Final Displacement Error, FDE)：
只以最後一個預測點與真實點的歐式距離作為整條軌跡的最後誤差



參考資料

- GAN參考資料：

- (GAN)<https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-ml-note-generative-adversarial-network-gan-%E7%94%9F%E6%88%90%E5%B0%8D%E6%8A%97%E7%B6%B2%E8%B7%AF-c672125af9e6>
- (GAN)https://ithelp.ithome.com.tw/articles/10196257?fbclid=IwAR0xUX_EAquigvAnO7cAVobU2gvIJg9DFLpsJxG95cPsCdnqzPVctl08yBQ
- (GAN應用場景) <https://ithelp.ithome.com.tw/articles/10207949>
- (Cycle GAN) <https://medium.com/hoskiss-stand/cycle-gan-note-bd166d9ff176>
- (Deepfake) https://technews.tw/2020/11/28/deepfake-2020-development-status/?fbclid=IwAR0zSwM_KoqAHTheXLtpwuzhtn7dDde0wxJgEYLTlt1xBRs260YUoQWk1F4

- GAN程式碼實作：

- https://github.com/daymos/simple_keras_GAN
- https://github.com/super13579/tensorflow-GAN-MNIST/blob/master/GAN_MNIST.py